MYDOS 3.Ø User Guide

Revision 3.Ø14

Charles Marslett

WORDMARK Systems
27Ø5 Pinewood Dr.
Garland, TX 75Ø42

February 18, 1985

CONTENTS

MYDOS 3.Ø User Guide

Revision 3.Ø14

Charles Marslett

I. INTRODUCTION

The disk operating system described in this manual is for the ATARI (Trademark of ATARI Corp.) 4ØØ 8ØØ, XL and XE series home computers. It is modeled after the first two Atari disk operating systems (DOS and DOS II) and may be considered an extension of the very "user friendly" concepts introduced with those operating systems. For this reason the system documented here is identified as MYDOS 3.Ø. The ATARI 81Ø disk drive is well supported by the DOS II operating system, but DOS II has only limited provision for a double density disk system (the ATARI 815 dual drive system), and no provision at all for a system that could change density dynamically (a "dual" density system). In MYDOS 3.Ø WORDMARK Systems has developed a disk operating system to allow for dynamic density changes and larger capacity disks even in a single drive configuration. It also supports the ATARI 1Ø5Ø double density format in a file system compatible with the original 81Ø disk operating systems.

II. SYSTEM REQUIREMENTS

MYDOS 3.Ø requires 16K of RAM (provided by most 4ØØ, and all 8ØØ or XL/XE series computers). It also requires one ATARI 81Ø compatible disk drive or disk controller, and if the dynamic density selection is used, the controller must provide the extended 81Ø interface described in Section 1Ø. If 1Ø5Ø double density is to be used, it must also support the 1Ø5Ø double density format command ($22) and the 1Ø5Ø status codes. Except for the 1Ø5Ø format double density, this extended interface is supported by the PERCOM dual density disk subsystems, the ATR8ØØØ disk/printer/RS232 controller manufactured by SWP, Inc.

Up to eight disk drives may be accessed, but only one is required. The resident part of the operating system leaves as much free memory as the ATARI DOS or DOS II do. MYDOS supports all documented functions of the DOS II operating system, so MYDOS 3.Ø supports most available software for the ATARI home computers. The BASIC and Assembler/Editor Cartridges, APX Pascal, the Deep Blue C Compiler, OSS BASIC A+, MEDIT and the AMAC macro assembler all run from double density disks with no alterations or limitations. ATARI BASIC has over 32,35Ø bytes left in a 4ØK system, even with eight double density disk drives configured. The memory occupied by the permanently resident part of MYDOS is $Ø7ØØ to $1DØØ and that part occupied

by the utility program exchanged with the MEM.SAV file extends to $3400. Unlike ATARI DOS, MEM.SAV need not be used if a program is to run directly from the disk (or cassette) and contains code located between $1D00 and $3400: the 'L' command will overwrite that part of memory. However, if the program is to be patched or run manually with the 'M' command, it must be loaded into MEM.SAV using the 'N' command because the menu program will be loaded back into memory on top of it before it can be modified or used!

The memory available to a program is affected only by the number of files to be open concurrently: each disk file that may be open at the same time requires 256 bytes of buffer space. Note that memory requirements are independent of the number of disk drives or the sector size (density), however.

III. MENU FUNCTIONS

The menu provided by MYDOS 3.0 identifies 18 common tasks that might need to be done. Rather than having to write a utility program (only a few lines of BASIC would perform most of the menu functions) or even remember the name and format of a DOS command, these tasks can be handled by entering a single letter. MYDOS 3.0 responds with a question asking for the details of the operation (which file, what density, "are you sure?" or whatever else it might need to know). After you enter the remaining information, the function is performed and another prompt is displayed.

You should notice two interesting things about the menu: the second line on the screen identifies the disk drives present on the system and what they appear to MYDOS 3.0 to be (single or double density and single or double sided). Following the drive number a letter 'D' identifies drives currently in PERCOM double density mode and the letter 'S' identifies those in single density or 1050 double density mode. If a disk is configured double sided, this is followed by an '=' and if it is configured single sided, the density is followed by a '-'. The next line describes the current default directory (that directory used when a disk is referenced by 'D:' (without a unit number after the 'D').

The second thing to notice is that after commands fill the screen, the menu 'rolls' off the top: some DOS programs keep the menu, MYDOS 3.0 does not. This permits more information to be shown on the screen when a long sequence of commands is needed to perform a function or when a directory is being listed or several files are being copied. To restore the menu to the screen, just type RETURN and the initial screen is restored.

Some commands require further information to prevent accidental damage to your disk files: the 'I' command and the 'J' command both require confirmation (through an additional key entry)

before distroying the destination disk. To abort either operation without damaging any existing disk files, simply press RESET or BREAK. The 'D', 'E', 'F' and 'G' commands (delete, rename, lock and unlock) all require an explicit file specification. All other commands assume the drive containing the default directory or all the files in the default directory (depending on whether the command affects an entire drive or a set of files).

Disk drive specifications and file specifications are made using the same rules: if only a drive is specified and file data is required, all files on that drive (or in the case of the 'K' 'L' and 'N' commands to save and load programs, the first file on that drive) will be the assumed choice. A drive is specified with a 'D:' (meaning the default drive), a number (with or without a trailing ':') or the capital letter 'D' followed by a number and a ':'. If you wish to specify the file or set of files to be referenced, the drive format must include a ':' or it must be omitted entirely --

EXAMPLES: D1:Test.obj, 1:TEST.ASM, or D2TEST (really D1:D2TEST) are valid file names, but d1:Test.obj or 1TEST.ASM are not.

The file name itself is either fully specified (refering to exactly one file on the disk) or includes wild cards (specifying a set of zero or more files). A fully specified file name consists of one to eight characters followed by a period ('.') and zero to three additional characters. The first character in the file name must be an upper or lower case letter, an underscore ('_') or the characters '@' or ''. The '' is the ATARI diamond graphic. The remaining character may be in that set or one of the digits 0-9. The 'wild card' characters are the characters '*' and '?': the character '*' or the sequence '*.' end either the 8 character or the 3 character field in the file name and match all possible characters. The character '?' matches any single file name character.

In addition to the main directory (containing up to 64 files or directories) each MYDOS 3.0 disk may also contain additional directories of 64 files each. If the main directory contained the directory file BAS and the file GRAPHIC1 were in the directory file BAS, it could be referenced with the filename, BAS:GRAPHIC1. If instead, GRAPHIC1 were in the directory GR.dir which in turn were in BAS, then the reference would be to BAS:GR.dir:GRAPHIC1 (and so on with as many names as needed). Because there is no limit to the number of directories on a disk (other than the buffer size of programs using the directories and number of available sectors on the disk), a single diskette can contain hundreds of files if necessary. Each directory is limited to 64 files or subdirectories though.

If a disk directory includes the files TEST.ASM, TEST.OBJ, TEST.C, TEST.ALM, TEASET.DOC, TRACE.FIL, and BETS.LST, the specification "t*.*" will not match any file name (since "t" and "T" are not the same letter to MYDOS 3.0). The

specification "T**" will match all but "BETS.LST" (since the others all begin with the letter "T"). The specification "?E??.*" will match the first four files and the last one (since the 8 character part of the file name must have no more than 4 characters in it and the second character must be an "E"). The specification "*?" will match only the file TEST.C (since it is the only file name with a single character in the 3 character field). The specification "????E**" will match the files TEASET.DOC and TRACE.FIL and none of the others (since the 8 character part of the file name must have at least 5 characters and the fifth must be an "E").

Where more than one file name is asked for, the first may be omitted by starting the response with a space or comma, and the last may be omitted by ending the line with a comma (the space cannot be used here since trailing spaces are ignored). If both file names are entered, they may be separated with either a space or a comma. Some commands may be modified using a letter following the character '/' after the file name (for example, 1/A or D1:TEST/A). The letter used (the modifier) generally means the same thing if it is allowed. Invalid modifiers are always ignored with no error indication at all. The modifier '/A' causes the results to be appended to the end of an existing file. This is applicable to 'C' (copy) and 'A' (directory) commands. The modifier '/N' causes the destination disk formatting to be skipped (saving about a minute) when used in the 'I' (initialize) and 'J' (duplicate disk) commands. It prevents the questions asked before changing each file if it is used in the 'D' (delete), 'F' (lock) and 'G' (unlock) commands; so we bend the rule only recently mentioned. In both cases, part of the function is skipped (see?). The '/X' command causes MYDOS 3.Ø to pause at the end of each read or write pass when copying data to allow you to change disks. This allows the writing of a directory of one disk onto another as a file (use the command 'A' followed by the entry "1,1:DRVØ1.dir/X") with only a single drive on the system. It also provides a method for dividing the entire contents of a large disk between several other disks by inserting one or another in the drive as each file is copied. The '/X' is assumed if only one file name is entered in the copy command. This emulates the operation of the ATARI DOS II 'O' command which is not implemented in MYDOS 3.Ø.

To omit copying files with extensions beginning with 'S', the '/S' modifier can be used in the 'C' (copy) command: for example, the line "2/S,1" will copy all files not matching the string *.S?? from drive 2 to drive 1.

A special modifier '/Q' is provided to premit selective copies of sets of files: a 'Y' response is required for each file to be copied, any other response to the copy prompt will result in that file being ignored.

IV. THE MENU COMMANDS

A. List a Directory or a Set of Files

The 'A' command will list the files on a disk with their sizes, followed by a line specifying the number of free sectors on the disk. If the line starts with an '*', the file has been locked and may not be modified or deleted without first being unlocked. A ':' before the file name marks those files that are directories. These files cannot be read or written as other files but only accessed as directories. No indication is made of the format of the file (ATARI DOS, ATARI DOS II, or MYDOS 3.Ø are the three supported file formats). See Section 6 for further information if you need more information about the files than the 'A' command provides. This command will list the directory information to the screen if only one file specification is entered. If two are entered, the second is taken as a destination file and will be overwritten (or appended to) with the directory data: the entry "1,P:" will write the directory of the disk on drive 1 to the printer, for example.

To list the files in a subdirectory, enter the name of the directory followed by the symbol ':'. For example, "1:TEST:BAS:" will list the files in the directory BAS which in turn is in the directory TEST in the main diskette directory. To list only those files with the extension ".MAC", you could enter "1:TEST:BAS:*.MAC".

B. Run the Cartridge

The 'B' command returns control to the cartridge in the first cartridge slot. If no cartridge is present, an error is displayed, and nothing happens. No additional information is required, so if a cartridge is present it is entered after loading MEM.SAV (if the last load command was an 'N') or immediately (if the last load command was an 'L').

C. Copy a File or a Set of Files

The 'C' command is used to make another copy of one or more files of data. The two file specifications asked for after entering the 'C' identify the source and the destination of the information being copied. Either may be fully specified disk file or a device specification (such as E:, P: or one of the RS232 ports R1: to R4:). The destination may be a set of disk files (specified with '*' and '?'s) only if the source specifies a file name for the destination to use. Copies from a file set to a device will implicitly write consecutive files to the device (generating a set of listings or a collection of cassette files for example). The source may be a set and the destination a single disk file, but unless the '/A' modifier is specified to append each copied file to the end of the previously copied files, only the last source file will remain on the destination disk.

Note that the 'C' command always uses the full memory space for a copy operation (unlike ATARI DOS II) and as a result, will always invalidate MEM.SAV if it is used. No pending program can be restarted after a 'C' or 'J' command.

D. Delete a File or a Set of Files

The 'D' command will remove all files that match the file specification entered asking for confirmation before each one is removed. This verification that the file is really the one to be removed can be disabled for the duration of this single 'D' command by adding the command modifier '/N' to the end of the file specification. In this case, all the matching files will be removed 'quietly' and the only further indication you will see is the prompt for the next command.

E. Rename a File or a Set of Files

The 'E' command changes the name of the source file or files to match the specification in the destination. Unlike other file specifications, the destination specification must consist of a single file name: it must not contain any directory names or a disk drive specification. For example, "D2:TEST:BASIC:NOTPNT.BAS,RANDIO.BAS" is the line entered to change the name of a file in the directory "D2:TEST:BASIC". To change the name of the directory "BASIC" to "ATBASIC", the line would look like "D2:TEST:BASIC,ATBASIC".

F. Lock a File or a Set of Files

The 'F' command limits access to the files identified. The files may not be deleted, renamed, added to or replaced without being first unlocked with the 'G' command. When a directory is listed, the files that have been locked using either the 'F' command or the 'lock' or 'open locked' functions provided through CIO will be marked with an '*' in the first column. The files that are locked may be read or loaded and executed normally, only modification or removal are prohibited.

Before each file is locked MYDOS asks you for confirmation with a message: for the file TEST, the message would read "Lock TEST?". Any answer but 'Y' will result in the file not being locked. The confirmation questions can be skipped by adding '/N' to the end of the file specification.

G. Unlock a File or a Set of Files

The 'G' command removes the limitations imposed on a file when it is 'locked' using the 'F' command. It does not alter the file or otherwise change the way the file is accessed or used. The same function may be performed in a program through the CIO

function to 'unlock' a file.

Before unlocking each file MYDOS asks for confirmation with a question that must be answered with a 'Y' if the file is to be unlocked; otherwise, no action is taken and the next confirmation question is asked. To disable the confirmation questions, enter '/N' after the file specification (see Section 4.4, on deleting files, for a more detailed explanation).

H. Write MYDOS 3.Ø to a Disk

The 'H' command is used to make a rebootable copy of the current MYDOS 3.Ø files in memory. The two files created or rewritten are 'DOS.SYS' and 'DUP.SYS'. 'DOS.SYS' is an image of the permanently resident file management routines accessed through CIO and the small interface package that loads and saves MEM.SAV (an image of the part of memory used to hold the nonresident part of MYDOS 3.Ø) and the second part of MYDOS 3.Ø itself ('DUP.SYS'). The file 'DUP.SYS' is a standard load file containing the part of MYDOS 3.Ø that is overwritten when a program is loaded into memory. Neither of these files is compatible with any other disk operating system either for the ATARI or any other home computer. Both should be treated as a single object. Never copy only DOS.SYS or only DUP.SYS to a disk without copying the other.

The files written to the disk by the 'H' command will reflect the configuration parameters currently in memory, which may be different from the ones active if the system were rebooted from disk again. (See Section 9 for the definition of the configuration parameters provided in the system and how to specify a modified configuration.)

I. Initialize a Disk

The 'I' command is used to prepare a new disk for use with the MYDOS 3.Ø operating system or to remove all the files on an old disk. The result of the 'I' command is a completely empty disk. The only data on the diskette is that system provided information defining the space available and the empty main directory. If the drive number is followed by a '/N' modifier, the diskette will not be reformatted, but just 'erased'. This is the recommended way to remove all the files on a diskette, rather than to use the 'D' command.

A disk is formatted to match the configuration (set up using the 'O' command and the 'P' command usually) by responding to the prompt: "Type 'Y' (or 'A') to FORMAT D?:" with a 'Y'. The response to format a 1Ø5Ø double density disk is 'A'. Note that only 1Ø5Ø drives and those supporting the 1Ø5Ø double density format command will properly handle the 'A' response.

J. Duplicate a Disk

The 'J' command copies all the information from one diskette to another. The information on the diskette is identified by the MYDOS 3.Ø bit map (on the VTOC sectors) if no sector range is specified. If a range is specified, the data to be copied are the sectors in that range and the VTOC is not examined. Specifying a sector range is done by adding two numbers seperated by a dash and enclosed in parentheses to the end of the drive specification(s). For example, to copy sectors 19 through 54 (tracks 1 through 3) from drive 1 to drive 3 the command line could be "1,3(19-54)".

The disk initialization done by the 'J' command is done without error checks: this means that a disk formatted with the 'J' command may have bad sectors (in the case of creating a backup disk, the disk will not be written to later so this is acceptable if no write errors occur). If the disk is to be a working disk, a more reliable approach is to initialize the disk (with the 'I' command) and then copy the data using the 'C' command. A two step operation, using the 'I' command followed by the 'J' command with the '/N' modifier is always required to back up ATARI 1Ø5Ø format double density disks.

If the destination disk is already a properly formatted MYDOS 3.Ø diskette, the '/N' modifier may be entered after either drive number to skip the formatting of the destination drive. Otherwise, the destination diskette will be formatted before the data from the source is copied to it. That is, either "1/N,2" or "1,2/N" will copy from drive 1 to drive 2 without first formatting the diskette in drive 2. To copy the first two tracks of a diskette without formatting the diskette being copied to, you could enter "1,2/N(1-36)" or "1 2/N(1-36)".

Note that the 'J' command, like that in ATARI DOS II, will use all of available memory to duplicate the diskette: this means that if memory has been saved using the MEM.SAV file, it will no longer be valid. Any pending program cannot be restarted after a 'C' or 'J' command.

K. Save Memory to Disk

The 'K' command builds a binary load file containing the data from the memory area specified, as well as an initialization and a run vector address if specified. If the file is not to execute an initialization routine on being loaded, the initialization vector should be omitted. If it is not to run on being loaded, the run vector should also be omitted (and trailing commas need not be typed in either). If either vector is entered as zero, that vector will not be invoked when the program is loaded. Note that the starting and ending addresses of the program and the initialization and run entry points of the program are all specified as hexadecimal numbers.

If MEM.SAV is active when the 'K' command is entered, the MEM.SAV file is loaded before writing the file to the disk.

L. Load Memory from a File

The 'L' command takes a binary load file from the disk and loads it into memory. The load file's initialization routine(s) will be executed and the program started at its run address unless the '/N' modifier is appended to the file name. This command disables the MEM.SAV file before loading and executing the program.

M. Run at an Address

The 'M' command is used to enter a program loaded without a run address, or to jump into any program without the need for a return address. It may be used to restart the computer (loading the AUTORUN.SYS file, if any) by specifying $E477 as the jump address. If MEM.SAV is active (enabled with the 'N' command and not since disabled by the 'L' command), the contents of memory will be restored from MEM.SAV before jumping to the address specified.

N. Load MEM.SAV from a File

The 'N' command takes a binary load file from the disk and loads it into memory. The load file's initialization routine(s) will be executed and the program started at its run address unless the '/N' modifier is appended to the file name. This command enables the MEM.SAV file before loading and executing the program and when control is returned to MYDOS the contents of memory will be saved in MEM.SAV.

If no file name is specified, the MEM.SAV file usage is enabled but no program is loaded or run (useful before entering a cartridge with the 'B' command if MEM.SAV is to be used when returning to MYDOS).

O. System and Drive Configuration

The 'O' command is used to specify the type (at least logically) of the disk drives on the ATARI computer. Additionally, it is used to specify the number of file buffers provided and to control verification after disk write operations. For owners of early production 800 and 400 computers with the ATARI A revision OS ROMs this command is used to disable the 'fast write' algorithms that do not reliably work with those computers. Also, by disabling fast writes, code in ROMs may be written to disk using the 'K' command (say to later be disassembled by any of several BASIC disassemblers). This is necessary since the fast write (or

"Burst mode I/O" as referred to in ATARI and OSS documentation) operations modify the text buffer, then restore it, during a write and this is obviously not possible if the text is in ROM. These three functions, which are not specific to individual drives, are selected by entering a RETURN when the prompt asking for a drive number is displayed. Three questions will then be asked: "Verify WRITEs" expects a 'Y' to be entered if all data written to the disk is to be read back to verify that it was not only written correctly, but that the data is in fact readable from the disk. The second question: "Number of File Buffers" expects a number, followed by a RETURN to specify the number of file buffers to be allocated. The third question, 'Fast WRITEs allowed?', should be answered 'Y' if the computer is an XL or XE series computer or a 400 or 800 computer with the B-revision OS ROMs. If the computer is an early 400 or 800 computer with the A-revision ROMs the answer should be 'N' to disable the fast write algorithm because the A-revision ROMs do not allow the sector buffer to be located anywhere in memory.

A RETURN does not retain the current value -- the result is YES as the answer to the 'Verify WRITEs' question, and 3 buffers as the answer to the number of buffers question. Anything but a 'Y' will enable fast writes in response to the third question.

If instead of a RETURN, a drive number or name had been specified then that drive would be reconfigured. The first question identifies whether the drive is to be included in system initialization (and thus be available for later use). If a non-existent disk is included it does not cause any problems with the system: it simply causes that disk to be examined each time the system is booted (adding perhaps a second to the time it takes to boot up MYDOS 3.0). If drive is excluded from the system, no further questions are asked. Otherwise, the second question asks if the drive is configurable: that is, is it like the ATARI 810 or 815 drives (with a fixed configuration) or is it like the PERCOM or ATR8000 drives. If the disk is not configurable it is assumed to be a 720 sector, single or double density ATARI 81x disk drive possibly supporting the ATARI 1050 double density format. Drives excluded from the system can be dynamically added by referencing them but they will always be treated as 5 1/4 inch 810 or 815 compatible drives (the default configuration).

If the first two answers are 'N' (do not exclude the drive) and 'Y' (it is configurable), the configuration is asked for: Is the drive double sided, how many tracks are there on the each side of the disk, and at what speed can it move the read/write head across the disk (what is its step rate). The first question is answered with 'Y' or 'N' ('Y' meaning 'yes' it is a double sided drive and diskettes formatted on it will be double sided). The second question is answered with 35, 40, or 80 followed by a RETURN if the disk drive is a 35 track, 40 track, or 80 track 5 1/4 inch floppy drive and with 77 if the drive is a standard 77 track 8 inch drive. No other number is accepted. The answer to this question specifies both the type of drive (8

inch or 5 1/4 inch) as well as the number of tracks per inch and total capacity of the drive. This answer is very important to the operation of the drive. The last answer is entered as a code: use the following table and the drive specifications to determine the proper value.

Code value	8 inch rate	5 1/4 inch rate
Ø	3 ms/track	6 ms/track
1	6 ms/track	12 ms/track
2	1Ø ms/track	2Ø ms/track
3	15 ms/track	3Ø ms/track

P. Disk Density Selection

The density used for most MYDOS commands is determined by the data written on the diskette and the operator need not worry about setting it. The 'P' command is provided to allow forcing the density setting for the format ('I') command.

MYDOS commands that access a diskette will automatically select the appropriate density, so the 'P' command will have no effect on the drive if any command accessing the drive configured with the 'P' command is executed before the format ('I') command.

Q. Create Additional Directories

When a diskette is formatted, an empty directory (the highest level or root directory) is created (empty). This directory is capable of holding up to 64 files or other directories. If additional directories are installed in this directory, each of the additional directories can contain up to 64 files as well. A directory is installed in an existing directory using the 'Q' command and responding to the question of what the directory name is with the name of the new directory. For example, if "TEST" and "BAS" are two directories in the root directory of the diskette in drive 1, "1:TEST:COMM" or "1:BAS:COMM" would both create a new directory in "TEST" and "BAS" respectively. "1:NEW:COMM" would not, however, since "NEW" does not already exist. A 'Q' command with the response "NEW" would create the first directory followed by an additional 'Q' command with the response "NEW:COMM" would create the two nested directories, though.

Each directory occupies 8 sectors and after it is created it may only be referenced as a directory (folowed by a ':' that is) or deleted. It may only be deleted if it is empty (if it has no files in it). A directory may be emptied by using the 'D' (delete) command and specifying the files "*.*/N" to remove all the files in the directory.

R. Set the Default Directory

The 'R' command is used to select a directory to be used when a file is referenced without the drive number: that is, when file names such as "TEST1.BAS" or "D:NEWCODE" or even ":BIGFILE" are used, they are assumed to be in the default directory. Programs run under MYDOS 3.Ø can access the contents of the current default directory by using a file name of the form "D:..." without the drive number explicitly entered. The default directory can also be set by calling the CIO function code 41 (set directory) routine.

The directory is set by inserting the diskette with that directory on it into the desired drive, then entering the file name of the directory with no trailing ':'.

If the diskette in the drive containing the default directory is replaced, the default should be redefined unless it is the root directory (this is because only the root directory is at the same location on all diskettes).

V. FILE MANAGER FUNCTIONS PROVIDED THROUGH CIO

MYDOS 3.Ø supports all CIO calls supported by ATARI DOS II, with some modifications to the OPEN (Function code 3) and the FORMAT (Function code 254) functions. Three additional CIO functions have been added: MAKE DIRECTORY (Function code 34), SET DIRECTORY (Function code 41) and LOAD MEMORY (Function code 39).

The OPEN function in ATARI DOS II does not use the data provided in the AUX2 byte, but in MYDOS 3.Ø the AUX2 byte contains three flags that control the file format and whether it will be created locked or not when the AUX1 byte is 8 (the file is opened for creation or replacement). If AUX2 bit 1 is set, the file will be written in the ATARI DOS (original) format unless the disk is double density (the original format is not viable for 256 byte sectors). When using double density disks, this bit is ignored). If AUX2 bit 2 is set, the file will be written in MYDOS 3.Ø format, and may contain sectors beyond absolute sector 1Ø23. Such a file may not (easily) be read using ATARI or OSS DOSs. If AUX2 bit 5 is set, the file will be written with the 'LOCKED' bit in the directory set initially. This is provided for use by multi-tasking functions (such as a print spooler, sequential file pre-reading function or other enhancements one might want to make to the standard OS or DOS provided functions).

Also, the opening of a file will result in the reestablishment of the disk density (so any prompt to change disks should be issued before opening the disk files -- a good idea in any case!).

The FORMAT function in ATARI DOS II does not provide for any variations to the standard disk usage: in MYDOS 3.Ø, the contents of the AUX1 and AUX2 bytes are used to specify the

number of sectors on the disk being formatted and whether the disk needs to be formatted by the controller as well as needing directory initialization. Bit 7 of AUX1 is set to skip the physical formatting of the entire disk surface when it is not required, and bits 6-Ø of AUX1 and all of AUX2 are used to specify the number of sectors on the disk being created (if all 15 bits are zero, the disk is assumed to be a standard 72Ø sector disk). The special case of formatting a 1-sector disk is used to force ATARI 1Ø5Ø double density formatting (if possible).

To load (and possibly execute) a program file, MYDOS provides the CIO function 39 call. From BASIC you can load and execute a program by executing the line: XIO 39, #3, 4, Ø, "D:MYPROG.OBJ". Any inactive IOCB can be used, and if AUX1=4 both the INIT and the RUN entries will be executed. If AUX1=5, the RUN entry will be executed, if AUX1=6, the INIT entry will be executed, and if AUX1=7, the file will be loaded without executing either entry point. Any other values of AUX1 is an error.

Another XIO call, XIO 34, has been added to create a directory. When a directory is created, the name used must not match any existing file or directory in its parent (for example if the directory to be created is named "D1:TEST:BUGS", there can be no other directory in the main directory named "TEST" nor a file named "TEST" there.

From BASIC the XIO 34 call is "XIO 34, #iocb, 8, Ø, dirname" where "iocb" is any available unit number, and "dirname" is the name of the new directory (with no trailing ':').

The final function added to those provided by ATARI DOS II is XIO 41, to define the default directory. The default directory is that which will be searched for a file if the file name begins with "D:". In ATARI DOS II this default directory is always "D1:" but in MYDOS 3.Ø, the default directory can be any root or subordinate directory on any disk in the system. The buffer address passed CIO in the XIO 41 call is the address of a string that contains the default directory name, terminated with either an end of line ($9B) or a null byte ($ØØ). The directory will be accessed before returning to the calling program so that an error in specifying the directory will be reported as early as possible.

VI. CIO FUNCTION CODES PROVIDED BY MYDOS 3.Ø

Function Code 3, OPEN

The open function uses the buffer address to point to an ATASCI string terminated with a character not Ø-9, A-Z, a-z, :, ?, or *. This string is the name of the file to be accessed or created. A good terminator for this string is either a null

($ØØ) or an end of line ($9B).

The AUX1 byte defines the usage of the file: 4 for input, 6 for directory data reading, 8 for creating/replacing output, 9 for creating/appending output and 12 for input/update (without extension). The AUX2 byte is used when a file is replaced or created, and contains three significant bits, bit 1 set causes a DOS I format file to be created if the diskette is single sided, single density (otherwise, it is ignored). Bit 2 set causes the MYDOS format to be used even if the diskette is a 4Ø track single sided diskette. And bit 6 set results in the file being LOCKed initially without an additional CIO call. In normal use, AUX2 is set to zero emulating ATARI DOS II usage. The length field is always ignored, and for input, update or directory access AUX2 is ignored.

Before finding the file in the directory, an open will result in an attempt to autoconfigure the density of the drive accessed and set the sector size accordingly.

Function Code 5, GET RECORD

The get record function reads a line of data into a buffer, the buffer being defined by its starting address and length. The line is defined as the data bytes in the file up to an end of line character ($9B) or until the buffer is full, whichever occurs first. The line is also terminated if the end of the file is read. All record I/O is buffered in MYDOS so record transfers are necessarily slower than unbuffered I/O.

No other fields of the IOCB are referenced or needed. Note that the ATARI ROM OS supports single byte I/O through the accumulator if the buffer length is set to Ø. In this case, GET RECORD and GET CHARACTERS function exactly the same way.

Function Code 7, GET CHARACTERS

The get characters function reads a fixed number of bytes from a file into a buffer, the buffer being defined by its address and length (two 16-bit number in the IOCB). The only case where the buffer is not always filled is if the end of the file is read. As is the case with get record calls, a single byte may be read into the accumulator by setting the length field to zero. A get character CIO call will be perform unbuffered I/O if the buffer is longer than 256 bytes (ATARI DOS II sets a similar threshold at 128 bytes). For this reason a single long input is considerably faster than several short ones.

Only the buffer address and length in the IOCB are used by the get characters function.

Function Code 9, PUT RECORD

The put record command will write a single line to an output file, the line defined the starting address of the buffer and either the length of the buffer if no end of line ($9B) bytes are encountered, or the first end of line byte. Only the buffer address and length in the IOCB are used in this command.

Function Code 11, PUT CHARACTERS

The put characters command will write the contents of a buffer defined by its address and length (in the IOCB), to a file opened for output. The entire buffer is always written to the file unless the write is to an output/update file and the end of the file is reached or the write is to an output/append or create file and the last sector on the disk has already been allocated. Only the buffer address and length fields in the IOCB are used when the put character function is used.

Function Code 12, CLOSE A FILE

To terminate use of a file (and for an output file, to write the incomplete buffer to the disk) the IOCB used to access the file should be closed. This is done by setting the function code in the IOCB to 12 and calling CIO. The close function does not use any of the data in the IOCB for any purpose whatsoever.

Function Code 13, READ STATUS

The read status command is issued to an unopened IOCB, with the buffer address that of a file name string. If the file is not present that error condition is returned, if it is locked, that error condition is returned; otherwise, a normal completion code is returned. Only the function code and the buffer address in the IOCB are needed.

Function Code 32, RENAME A FILE

The rename function is passed a character string (pointed to by the buffer address in the IOCB), and the first part of the string is a file name string identifying the file or files to be renamed. Following a single invalid character (one invalid in the file name, that is) another file name must be in the string: this second file name cannot include any drive or directory names. An example, using a comma as the invalid character, is "D2:TEST:PGMS:A.OUT,ZCPY" which will change the string needed to access the file "D2:TEST:PGMS:A.OUT" to "D2:TEST:PGMS:ZCPY" -- Note that only the last file name (if subdirectories are used) can be changed, to change "PGMS" to "MLPROGS", the buffer must contain "D2:TEST:PGMS,MLPROGS" and the rename will also change the full names of all files in "D2:TEST:PGMS" (to belabor the obvious).

Like the open function, the rename function will attempt to adapt to the density of the currently mounted disk in the specified drive.

Function Code 33, DELETE A FILE The delete function removes any files that match the file name string pointed to by the buffer address in the IOCB. Files locked will not be deleted, so must be unlocked before being removed, and directories that are not empty (that have a file, even an empty file, in them) cannot be deleted. If either case is attempted, the corresponding error code is returned. Otherwise, the files are removed and their data areas are returned to the free space on the disk. Like other Atari DOSs, in MYDOS files removed cannot be recovered after being deleted. This is unlike some other operating systems that preserve deleted files for as long as practical before overwriting them.

Like the open function, the delete function will attempt to adapt to the density of the currently mounted disk in the specified drive.

Function Code 34, MAKE DIRECTORY

The make directory function will create a new subdirectory on a disk (it is not used to create the first directory, that is the "root directory" identified by the drive specification "D1:", for example). It is called through CIO by storing the address of the new directory's name in the IOCB buffer address and setting up AUX1 and AUX2 as for an open call (see Function code 3), normally AUX1=8 and AUX2=Ø. This function has no effect on the current default directory, and if it is desired to make the newly created directory the default one, the program must make a set directory call (Function code 41) following the make directory call (the order is very important, because the default directory cannot be set to a nonexistent directory).

Like the open function, the make directory function will attempt to adapt to the density of the currently mounted disk in the specified drive.

Function Code 35, LOCK FILE

A file can be "locked" so that it may not be modified or deleted inadvertently by calling CIO with the lock function. The buffer address is used to point to a file name string that identifies the files on the disk to be locked. The only file modification that can be performed on a locked file is to unlock it. The lock function can be requested for a file already locked, and it will return no error (unlike other file modification calls to CIO), but the status of the file will not have been changed either. Like OPEN, the drive density will be adjusted before accessing the directory.

Function Code 36, UNLOCK FILE

The unlock function is identical to the lock function except that is reenables the modification or deletion of an unlocked file. A file that is not locked can be unlocked with no error returned and no change in the file's status. Like OPEN, the drive density will be adjusted before accessing the directory.

Function Code 37, POINT TO POSITION IN A FILE

The point function is passed the 3-byte disk address to be positioned to in the twelvth through fourteenth bytes of the IOCB. On return, the next byte read from that IOCB will be the one that was read or written next after the corresponding note function was executed. A point call to CIO can only be made if the file can be used for input: that is, if it is opened for input or update processing. The first two bytes of the disk address are a sector number (in low byte/ high byte format) and the third is the byte within the sector.

Function Code 38, NOTE POSITION IN A FILE

The note function returns in the twelvth through fourteenth bytes of the IOCB a 3-byte disk address that may be used at a later time to reposition the file using the point function. The note function can be used on files open for input, output, update or appending. The three bytes returned are the low byte of the sector address, the high byte of the sector address, and the byte within the sector in that order.

Function Code 39, LOAD MEMORY

The load memory function takes a file formatted in the ATARI DOS II executable program format (generated by the "K" command, by the assembler/editor cartridge, by AMAC or MAC65, or by any of several compilers for the ATARI computers) and loads its contents into the computer's memory as specified in the file. No offset control is provided and no part of memory is protected from the loading process. The initialization and execution addresses (if any) can be individually enabled and disabled, however, to permit loading and patching a program then writing it back to the disk for normal use.

To load a program into memory, the address of the file name string is stored into the buffer address and a value of 4, 5, 6 or 7 is stored into the AUX1 field. If AUX1 is 4, both the initialization routines and the run address are executed after closing the IOCB used but before returning to the calling program. If AUX1 is 5, the initialization routines are disabled, but the program will be run. If AUX1 is 6, the initialization routines will be run, but the program execute address will be loaded and ignored. If AUX1 is 7, the text of

the program will be loaded into memory, but no other activity will be performed. Like OPEN, the drive density will be adjusted before loading the file.

Function Code 41, SET DEFAULT DIRECTORY

The set directory command will use the contents of the buffer as a file name, determine the disk density and open the specified file, determining if that file is a valid directory. If so, it will become the new default directory. That is, file names of the form "D:..." will be assumed to be in the default directory (which may be on any disk in the system and may be either the root directory of that disk or a subdirectory).

Only the buffer address and the function code are significant when setting the default directory.

Function Code 254, INITIALIZE A DISK

The initialize function uses the contents of the buffer pointed to by the buffer address to identify the drive containing the diskette to be formatted. If AUX2 bit 7 is zero, the disk is formatted according to the capacity data in the system control table defined using the 'O' and 'P' commands. In this case, if two attempts fail to format the disk perfectly (with no bad sectors), all bad sectors are marked as used and the disk will have fewer free sectors than expected but it will be usable. The return status will be positive and one greater than the number of bad sectors removed from free sector table.

If AUX2 bit 7 is set to 1, the format operation is skipped and an empty file system is written to the diskette (This assumes the disk is preformatted).

The remaining 15 bits of AUX1 and AUX2 are used as a 15 bit number to specify the number of sectors available on the disk (permitting the use of the last few sectors of a disk outside the file system if desired). If a size of one sector is specified (AUX1 = 1, AUX2 = Ø), the disk drive is assumed to be an ATARI 1Ø5Ø drive and the disk is formatted as a 1Ø5Ø double density disk with 1Ø4Ø sectors using the 1Ø5Ø double density format code $22.

VII. DISK STRUCTURES SUPPORTING MYDOS 3.Ø

MYDOS 3.Ø uses the first three sectors of a disk to hold some disk information and the initial boot program if the drive contains DOS.SYS and Dup.SYS. Sector $168 (and sector $167 if the disk is formatted as a high capacity disk) is used to hold a bit map of available sectors and several flag byte identifying the default format of files on the disk. Sectors $169 through $17Ø contain main disk directory data, identifying

the files on the disk, their sizes and their starting sector number.

Note that this usage, when the disk is declared to be DOS II compatible, is in fact exactly the same as ATARI DOS II would make of the disk. The only significant change made when the high capacity format is chosen are that sector $167 is also reserved for use by the system. The high capacity disk directory may be read by ATARI DOS II, but the data in the files can only be accessed if it falls in the first 1023 sectors of the disk and then only if the file number checking code in DOS II is disabled. This format and MYDOS 3.0 support accessing disks of up to 4015 sectors of 256 bytes each.

Compatibility is further reduced if subdirectories are used: to ATARI DOS 2.0, the subdirectories will appear to be simple files with unreadable contents. The subdirectory's files will not be accessable and the subdirectory can be damaged if it is written to (even by appending). For this reason disks sold to the general public, exchanged with friends, and so forth, should not contain subdirectories unless there is reason to require that the disk be used with MYDOS. A further problem with exchanging diskettes is that each vendor of double sided disk systems for the ATARI is using a different format for the second side -- for this reason, double sided disks not only require both computers use MYDOS, but also require that they use the same disk system (PERCOM, SWP, Astra, Indus or Rana).

VIII. MYDOS 3.0 MEMORY MAP

The MYDOS 3.0 disk operating system occupies the area from $0700 to $1D00 at all times, and when the menu is active, it also occupies the area from $1D00 to $3400. In addition, the first 16 bytes of the floating point workspace ($D4 - $E3) are used by MYDOS 3.0 at that time. Unlike ATARI DOS II, MYDOS 3.0 also calls the floating point ROM entry points.

IX. RDOS 3.0 MEMORY MAP

The RDOS 3.0 disk operating system occupies the area from $0700 to $0BE0 at all times, and when the menu is active it also occupies the area from $0BE0 to $2000. As does MYDOS 3.0, it uses the floating point ROM entry points and 16 bytes of the floating point ROM workspace from $D4 to $E3.

X. CUSTOMIZING A SYSTEM DISK

Number of Files Open at Once

The number of files that may be simultaneously open is set with the same byte as in ATARI DOS II: location $0709 (decimal

18Ø1). This byte contains a number from Ø to 16 setting the number of disk files that may be open at the same time. Normally it is set to 3, the smallest number that supports all the functions in the MYDOS 3.Ø menu. Specifically, a copy from one disk file to another requires three open disk files. The value in the distributed version of MYDOS 3.Ø is three, to permit more or fewer files, use the "O" command followed by a RETURN. To permanently change the maximum number of open files, modify the value using the "O" command then use the 'H' command to write a modified MYDOS 3.Ø system to a disk.

Controlling the Disk Drives Accessed when Booting

Like ATARI DOS II, MYDOS 3.Ø automatically identifies the disk drives that are present when booted up initially and any time it is reinitialized (some programs do this on exiting to the DOS and it is always done if the RESET key is pressed. MYDOS 3.Ø is distributed with drives 1 and 2 configured, all others are omitted in order to speed up the booting process. To modify the maximum configuration MYDOS 3.Ø will use, invoke the 'O' command for each drive to be added to (or removed from) the system. Pressing the RESET key will then use this value to redefine the system. To permanently change the maximum drive configuration, use the 'H' command, writing a new copy of MYDOS 3.Ø back to the system disk.

Enabling or Disabling Write-with-Verify

MYDOS 3.Ø is distributed with all data written to the disk read back to verify that it was not only written to the diskette correctly, but that there was not a problem (dust, a scratch in the oxide coating, or some other problem that may have arisen since the diskette was formatted) that prevented the data from being read back from the diskette. If the programs being run have no long term value (games for example, often write daily high scores to the disk, and loss of such data might result in a few screams or moans, but so does waiting for a slow update of the scores after each game!).

This byte is defined, along with the count of the number of buffers to be allocated when the file manager is initialized, whenever the 'O' command is invoked with no drive specified (only a RETURN is entered in response to the drive number query). To permanently alter it, rewrite MYDOS back to the disk using the 'H' command after changing it.

XI. DISK DRIVE INTERFACE (via SIO)

The physical disk drives and diskettes are external to the ATARI home computers and the ones supported by MYDOS 3.Ø are normally attached to the "serial interface connector" on the right side of the computer. The software in the operating

system (OS ROMs) to access the devices attached to that connector is call the "serial I/O driver" or SIO for short. The MYDOS 3.Ø disk operating system uses this lower level driver to pass all commands and information to and from the physical disk drive. Several commands were defined by ATARI to communicate with the 81Ø disk drive and both PERCOM and SWP, Inc., have adopted a slightly extended version of this set of commands. MYDOS 3.Ø will operate in a limited fashion with any disk system that supports the entire 81Ø set, but the full set of commands is required to support all the functions.

The minimum set of disk drive functions to support MYDOS 3.Ø (or ATARI DOS II for that matter) are:

Device	Unit	Command	Direction	Byte Ct.	Aux. Bytes	Function
$31	Drive#	$21	From Drive	128/256	1 to 72Ø	FORMAT DISK
$31	Drive#	$5Ø	To Drive	128/256	1 to 72Ø	WRITE(no vfy)
$31	Drive#	$52	From Drive	128/256	1 to 72Ø	READ
$31	Drive#	$53	From Drive	4	1 to 72Ø	READ STATUS
$31	Drive#	$57	To Drive	128/256	1 to 72Ø	WRITE(verify)

The byte count is always 128 for a small sector drive, and is 128 for the first three sectors (1, 2, and 3) of a large sector drive. All other sectors on a large sector drive are 256 bytes long.

The FORMAT function is never called with a sector number not in the range of 4 to 72Ø. It expects 128 bytes from a small sector drive and 256 bytes from a large sector drive.

The first byte returned by the READ STATUS command is expected to indicate the sector size -- if bit 5 is a 1 (bit 7 is the sign bit) then the sectors are large (256 bytes), otherwise, they are small (128 bytes). Bit 6 is used to identify an ATARI 1Ø5Ø double density disk -- if set, the disk is assumed to be formatted as 1Ø4Ø 128-byte sectors.

The auxiliary bytes are treated as an address to a sector on the diskette, and range from 1 to 72Ø (when in DOS II compatible mode) or from 1 to 4Ø15 (when accessing large capacity disk drives).

The additional functions needed to support all the features of MYDOS are:

Device	Unit	Command	Direction	Byte Ct.	Aux. Bytes	Function
$31	Drive#	$22	From Drive	128	1 to 72Ø	FMT. 1Ø5Ø-D
$31	Drive#	$4E	From Drive	12	1 to 72Ø	READ CFG.
$31	Drive#	$4F	To Drive	12	1 to 72Ø	WRITE CFG.

The first command ($22) permits formatting a disk in ATARI 1Ø5Ø double density format. The other two ($4E and $4F) support the 'P' command, permitting reconfiguration of a disk drive on demand: to format a diskette, for example. They also support identification of a disk drive as an 8 inch or double sided drive. The individual bytes are defined as follows:

byte Ø: Tracks per side (4Ø for a standard disk drive)
byte 1: Disk Drive Step Rate (as defined by Western Digital)
byte 2: Zero (high byte of sectors/track)
byte 3: Sectors/Track (18 for standard diskettes)
byte 4: Side Code (Ø=single sided, 1=double sided)
byte 5: Disk Type Code --
bit 2: Ø=single density, 1=double density
bit 1: Ø=5 1/4 inch diskette, 1=8 inch diskette drive
byte 6: High byte of Bytes/Sector (Ø for ATARI 81Ø compatible)
byte 7: Low byte of Bytes/Sector (128 for ATARI 81Ø)
byte 8: Translation control
bit 7: 1=4Ø trk. disk I/O on an 8Ø trk. drive
bit 6: Always 1 (to indicate drive present)
bit 1: 1=Handle sectors 1, 2, and 3 as full size sectors
bit Ø: 1=Sectors number Ø-17 (for example) not 1-18
bytes 9-11 are not used (and should be zero)

An additional requirement is that the disk drive automatically switch density if required to read sector 1. This is necessary if one is to 'boot' either single or double density disks and is used by the MYDOS 3.Ø system to automatically set the drive density.

3	Last byte of file read, next read will return EOF (MYDOS)
2-127*	Format resulted in (n-1) bad sectors marked (MYDOS)
128	Break Abort (OS ROMs)
129	IOCB already open (OS ROMs)
130	No such device defined in the system (OS ROMs)
131	Write-only IOCB, cannot read (OS ROMs)
132	Invalid command (OS ROMs)
133	Device or File not open (OS ROMs)
134	Invalid IOCB reference (OS ROMs)
135	Read-only IOCB, cannot write (OS ROMs)
136	Attempt to read past end of file (MYDOS)
137	Truncated record (OS ROMs)
138	Device Timeout (OS ROMs)
139	Device NAK (serial bus failure, OS ROMs)
141	Cursor out of range for graphics mode (OS ROMs)
142	Data frame overrun (serial bus failure, OS ROMs)
143	Data frame checksum error (serial bus failure, OS ROMs)
144	Device I/O error (in peripheral hardware, OS ROMs)
146	Function not provided by handler (OS ROMs)
147	Insufficient RAM for graphics mode selected (OS ROMs)
160	Invalid Unit/Drive Number, zero or greater than 8 (MYDOS, OS ROMs)
161	No sector buffer available, too many open files (MYDOS)
162	Disk full, cannot allocate space for output file (MYDOS)
163	Not used: Atari DOS II system I/O error code
164	File number in link does not match directory location (MYDOS)
165	Invalid file name (MYDOS)
166	Byte not within file, invalid POINT request (MYDOS)
167	File locked, cannot be altered (MYDOS)
168	Invalid IOCB (MYDOS, OS ROMs)
169	Directory full, cannot create a 65-th entry in a directory (MYDOS)
170	File not in directory, cannot be opened for input (MYDOS)
171	IOCB not open (MYDOS, OS ROMs)
172*	File or directory of same name already exists, cannot create (MYDOS)
173	Bad diskette or drive, cannot format diskette (MYDOS)
174*	Directory not in parent directory (MYDOS)

175* Directory not empty, cannot delete (MYDOS) 180* Invalid file structure for loading memory (MYDOS) 181* Invalid address range for loading memory, END<BEGIN (MYDOS)

* -- New error codes, not present or different in Atari DOS II.

Most error codes are identical to those returned from ATARI DOS II, the differences result from the expanded capabilities of MYDOS 3.0. Specifically, Error 164, indicating a file number mismatch, only occurs if the file is written in DOS II or ATARI DOS I format. Errors 180 and 181 can only occur when XIO 39 is invoked to load a file, the ATARI DOS II equivalent function returned a code in the X-register. Errors 172 and 175 apply to

creating and deleting directories and have no ATARI DOS II equivalent, and Error 174 applies to accessing files in subdirectories, so it also has no ATARI DOS II equivalent. Error code 173 serves the same function as it did in ATARI DOS II, but is returned more often (identifying bad diskettes more reliably), and if the disk is usable, a count of bad sectors (plus 1) may be returned.

MYDOS 3.1 User Guide

Revision 3.19

Charles Marslett

I. The ATR8000 Serial I/O Handler

This document is an appendix to the MYDOS 3.0 User Guide and is not intended to be used alone. It describes the functions provided by MYDOS 3.1 that are not provided users of MYDOS 3.0. One significant difference is the fact that MYDOS 3.1 occupies memory up to address $2100 and the utility or menu program extends to $3800.

II. Programming Serial I/O Activity

When MYDOS 3.1 is booted, the RS232 serial port handler and the disk I/O handler are linked into the Central I/O (CIO) tables. The disk I/O handler is accessed using the CIO function calls described in Section 6 of the MYDOS 3.0 User Guide. Those used to access the RS232 serial port handler are described in following subsections. As with the ATARI 810 and 850 peripheral devices, the disks are refered to when the device name begins with "D" and the serial port is refered to when the name begins with "R".

Most programs written for use with the ATARI 850 interface will function properly with the ATR8000 and MYDOS 3.1. There are several limitations to this compatibility, however. Software on protected diskettes will not run with MYDOS 3.1 since it includes its own RS232 and disk I/O drivers. There is no simple way to replace the supplied I/O handlers, and if they are, the MYDOS functions may or may not reliably support the program. Programs that may be run with the standard distributed version of Atari DOS 2.0 will in most cases run correctly (again, if non-standard entries to operating system routines are made by the program, MYDOS will probably not operate as Atari DOS 2.0, OS/A+ or DOS XL. Further, programs that use two or more serial ports of the Atari 850 will not run properly because the ATR8000 has only one such port. I/O directed to or from any one of the four ports will be handled through the single port of the ATR8000 and there is no way to determine the real destination or source of the serial data stream. The status returned by the MYDOS 3.1 serial handler is somewhat different from that returned by the ATARI 850 handler: a result of the differences in the control lines provided by the ATR8000 and the inability of the MYDOS 3.1 serial driver to handle block mode output. MYDOS 3.1 does not support 5, 6 or 7 bit serial data (including parity if used) so programs for Baudot or

Correspondence codes will not run properly. Lastly, the transmit and receive buffers used by the MYDOS 3.1 serial handler cannot be replaced by user supplied ones: if the program provides its own buffers they will be ignored. The dedicated buffers provided by MYDOS 3.1 are each 256 bytes long.

Use of the serial port will also interfere with the operation of the disk I/O timeout and the printer spooling activities. While concurrent I/O is enabled, the printer will not print any text in the spooler buffer and the disk will be turned off immediately. When the serial port is closed, printing will be resumed (with no lost data) and, of course, disks may again be accessed.

Function Code 3, OPENRS232 PORT

This function uses the AUX1 byte and the buffer address (pointing to the device name: 'R:'). The values permitted for the AUX1 byte are Ø-15: if bit 2 is on, input is permitted, if bit 3 is on, output is permitted. Bits Ø and 1 are ignored: bit Ø will normally be set to enable concurrent I/O when accessing an Atari 85Ø. The ATR8ØØØ driver always uses concurrent I/O and ignores the state of this bit. Bits 6 and 7 must be zero. Opening the RS232 port does not activate concurrent I/O or transfer any data over the serial bus (it is entirely internal to the ATARI computer).

Function Code 5, GET RECORD

The get record function transfers data from the input buffer area to the memory pointed to by the buffer address in the IOCB until an EOL character is transferred. If translation is active this is an external carriage return (CR) character. In no case, however, will more than the number of bytes in the length field of the IOCB be transferred.

Function Code 7, GET CHARACTERS

The get characters function is identical to the get record function except that the EOL or CR character is ignored. The call will always transfer the number of bytes specified by the length field.

Function Code 9, PUT RECORD

The put record will transfer the number of byte specified by the length field or until an EOL character is encountered in the buffer pointed to by the buffer address in the IOCB. The data is moved to the RS232 buffer and if sufficient room is available to contain the record the calling program will

continue executing before the data is actually transmitted.

Function Code 11, PUT CHARACTERS

The put characters function transfers the number of bytes specified by the length field in the IOCB from the buffer pointed to by the buffer address into the transmit buffer in the handler. Except that it treats EOL characters as it does all others, the put characters function is identical to the put record function.

Function Code 12, CLOSE RS232 PORT

Issuing the close function call to the RS232 serial port handler will cause the calling program to wait til all text in the transmit buffer is transmitted, then it disables concurrent I/O activity, marks the IOCB free, and flushes (throws away) the data in the input buffer. This is the only 850 compatible way to terminate concurrent I/O.

Function Code 13, READ STATUS

The read status command is used to obtain information about the serial I/O port and the buffers used to access it. The command may be issued either when concurrent I/O is active or when it is not. Note that the pin numbers in the following tables refer to the RS232 25-pin D-shell connector assuming the jumper block in the ATR8000 is configured for a modem and the SWP provided ribbon cable used to connect to a standard modem is used.

If concurrent I/O is inactive, two bytes of status are returned: the first (Error Byte) is defined as follows:

Bit	Value	Meaning
7	128	Data framing error (RX)
6	64	Data byte overrun (RX)
5	32	Data byte parity error (RX)
4	16	Receive data buffer overrun
2	4	Carrier detect (pin 8) is required and is not ready

The second (Line Status Byte) is defined according to the following table.

Bit	Value	Meaning
6,7	192	Carrier detect (pin 8) or Ring Indicate (pin 22) is on (which one is selectable using the "O" command in the MYDOS 3.1 menu)
4,5	48	Ring indicate (pin 22) is on
2,3	12	Carrier detect (pin 8) is on
0,1	3	Data in is high (pin 3)

If the concurrent I/O is active on the channel, four data bytes are returned: the first is the same as before, the second is the number of bytes in the receive buffer awaiting processing. The third byte is always zero and the fourth is the number of bytes in the transmit buffer waiting to be sent over the serial channel.

Function Code 34, CONTROL SERIAL PORT OUTPUTS

The two lines in the RS232 serial port controlled by the ATR8000 are under direct program control using the control outputs function. However, if concurrent I/O is active, only the data out (TD) line can be altered. The IOCB may be a free one or it may be opened to the serial handler and only the AUX1 byte is used by the handler. Bit 7 or bit 5 enable changing the DTR control line and bit 1 enables changing the data out (TD) line. Bits 6, 4 and 0 are the corresponding new values to set the line(s) to. Note that if bit 7 is set, bit 6 controls the state of the DTR line, if bit 5 is set, bit 4 controls the state of the line. If both are set and bits 6 and 4 differ, the state of the output line is indeterminate.

Function Code 36, SET BAUD RATE AND CONFIGURE PORT

The baud rate, number of stop bits, and control of status line checking are set using the CIO function 36. In this function, bit 7 of AUX1 on results in transmitting 2 stop bits per byte (off results in one stop bit). Bits 4, 5 and 6 should be zero and bits 3-0 set the baud rate according to the following table:

Bits 3-0		Bits/Second
0	0000	300
1	0001	45.5
2	0010	50
3	0011	56.875
4	0100	75
5	0101	110
6	0110	134.5
7	0111	150
8	1000	300
9	1001	600
10	1010	1200
11	1011	1800
12	1100	2400
13	1101	4800
14	1110	9600
15	1111	9600

If AUX2 is zero, the carrier detect line (pin 8) need not be asserted before starting concurrent I/O with an XIO 40, a read or a write command. If AUX2 is between 1 and 7, carrier detect must be asserted by the modem (or whatever) before any communication over the RS232 port is allowed.

Function Code 38, SET TRANSLATION AND PARITY

The contents of the AUX1 byte are interpreted according to the following tables to set translation modes and to control outgoing parity. It also controls the interpretation of incoming parity bits (bit 7 of each byte). The contents of AUX2 are substituted for any untranslatable byte if heavy translation is enabled.

Value to Add	Means:
Ø	When transmitting a carriage return byte (CHR$(13)), do not add a line feed character (CHR$(1Ø)) to the transmitted data.
64	Append a line feed character after each carriage return transmitted.
Ø	Translate EOL (CHR$(155)) to CR (CHR$(13)) and zero bit 7 of all other characters transmitted
16	Also convert $ØØ-$1F to CHR$(AUX2)
32	Do no translation (send the data exactly as presented)
Ø	Ignore and do not change parity
4	Check for odd parity and clear bit 7
8	Check for even parity and clear bit 7
12	Ignore parity and clear bit 7
Ø	Leave bit 7 unchanged when transmitting data
1	Set output parity odd
2	Set output parity even
3	Set output bit 7 to 1

Function Code 4Ø, ENTER CONCURRENT I/O MODE

This function enables the serial I/O port handler and starts up the buffer handling routines. Unlike the ATARI 85Ø function, it cannot be used to specify an alternate input buffer. Also, unlike the Atari 85Ø function, it need not be specifiec at all: the first read or write accessing an open RS232 port will automatically enter concurrent mode I/O.

Also different from the Atari 85Ø, concurrent mode I/O use of the bus will terminate on any standard CIO bus transaction. This will leave the handler in a state of disrepair, but the contents of RAM can be written to disk or otherwise salvaged.

Because this function is not required to start an input function, the BASIC 'ENTER' and 'LOAD' functions can be used with the ATR8ØØØ serial port. As with the ATARI 85Ø, BASIC's 'LIST', 'SAVE' and 'LPRINT' can also be used.

III. RS232 Handler Error Codes

Error Code	Meaning

130 Not returned by the ATR8000 RS232 driver in MYDOS (The Atari 850 returns this code if port code is not R1, R2, R3 or R4)

131 Read function not enabled in OPEN and a GET or INPUT was attempted

132 Invalid XIO function code (possibly caused by doing block mode output)

133 IOCB not open and a read, write or enter concurrent I/O command (XIO 40) was attempted

135 Write function not enabled in OPEN and a PUT or PRINT was attempted to the serial port

138 The Atari OS ROMs attempted to perform the physical transfer of data to or from the ATR8000 and was not able to detect the pre- of the ATR8000 (check cables and power switches)

139 Carrier detect was not asserted when the RS232 port activity was requested (a read, write or enter concurrent I/O) -- Must be enabled using the AUX2 field of the XIO 36 command.

151 Not returned by the ATR8000 RS232 driver in MYDOS (The Atari 850 returns this code when concurrent mode I/O is attempted and has not been provided for in the OPEN call)

152 Not returned by the ATR8000 RS232 driver in MYDOS (The Atari 850 returns this code if the user supplied buffer description is not valid)

153 Returned if concurrent mode is requested (by an XIO 40 function call), and the RS232 port is already in concurrent mode

154 Not returned by the ATR8000 RS232 driver in MYDOS (The Atari 850 returns this code if a read or write is requested before issuing an XIO 40 request, and concurrent I/O was specified in the OPEN request)

IV. Using the ATR8000 Serial Port with the Atari 850

The ATR8000 serial port can be redefined to use any undefined I/O device name using the following simple program. This will permit you to use both the Atari 850 and the ATR8000 serial port by booting up with MYDOS 3.1 and including the Atari 850 AUTORUN.SYS file on the boot disk (at the expense of having two serial I/O drivers in memory, however). After running the following program, the DOS menu should be displayed, write the MYDOS files to the boot disk and whenever that disk is used to load MYDOS into memory, the RS232 device will be referred to as "Q:".

```
100 FOR I=7*256 TO 48*256
120 IF PEEK(I) <> ASC("R") THEN 140
```

MYDOS 3.1 User Guide

Revision 3.19

Charles Marslett

WORDMARK Systems
2705 Pinewood Dr.
Garland, TX 75042

February 18, 1985

CONTENTS